COSE322 System Programming Assignment #1

Instructions

- **Deadline:** October 6th, 2024, by 11:59 PM.
- Carefully read the description of each problem and write the corresponding code.
- Create a separate Cargo package for each problem, with the following package names:
  - Problem_1, Problem_2, and Problem_3
- Write a report explaining how you solved each problem. The report for each problem is limited to two A4 pages.
- Save the report as a PDF file, named in the following format:
  - Report_[Your student ID].pdf (e.g., Report_2024210046.pdf)
- Compress your three Cargo packages and report into a single .zip file and submit it on Blackboard. The .zip file should be named as follows:
  - Assignment1_[Your student ID].zip (e.g., Assignment1_ 2024210046.zip)
- **Important:** If the submission format (file name, type, compression, etc.) is not followed, points will be deducted without exception
- **Performance requirement:** Your algorithm should not have excessively high complexity. Each problem's execution should be completed within approximately 1-2 seconds.

Problem 1. Minimum Grade

Given the minimum GPA criteria for this semester and the grades of all subjects taken except one, output the minimum letter grade required for the remaining subject to meet the minimum GPA criteria. The grade conversion table according to the scores is as follows:

Letter	A+	A0	B+	B0	C+	C0	D+	D0	F
Grade	4.50	4.00	3.50	3.00	2.50	2.00	1.50	1.00	0.00

The GPA for this semester is calculated as $(the\ sum\ of\ (credits \times grade\ of\ each\ subject)\ over\ all\ subjects) \div (total\ credits\ taken\ this\ semester)$, rounded down to the third decimal place. If the GPA for this semester **exceeds** the minimum GPA criteria, it is considered to have met the criteria. However, **be careful of floating-point errors when using floating-point types**.

Input

- The first line contains the number of subjects taken this semester N and the minimum GPA criteria X separated by a space ($2 \leq N \leq 24;\ 0.00 \leq X \leq 4.50;\ X$ is a real number given up to two decimal places).
- From the second line to the $N-1$th line, the credits c_i and the letter grade g_i of each subject are given separated by a space ($1 \leq c_i \leq 6;\ g_i$ is one of A+, A0, B+, B0, C+, C0, D+, D0, F).
- The next line gives the credits L of the remaining one subject ($1 \leq L \leq 6$).

Output

- Output the minimum letter grade required for the remaining subject to meet the minimum GPA criteria.
- If it is impossible to meet the criteria regardless of the grade received, print "impossible".

Example 1

Input	Output
5 3.59 4 A+ 3 B+ 3 C+ 1 D0 3	A+

If the grade for the remaining subject is A+, the GPA for this semester would be $((4 \times 4.5) + (3 \times 3.5) + (3 \times 2.5) + (1 \times 1.0) + (3 \times 4.5)) \div (4 + 3 + 3 + 1 + 3) = 3.60714285$ ... rounded down to the third decimal place, 3.60.

Example 2

Input	Output
3 4.44 5 A+ 4 A+ 1	A0

Example 3

Input	Output
5 2.54 3 B+ 2 B0 2 C+ 2 C0 1	F

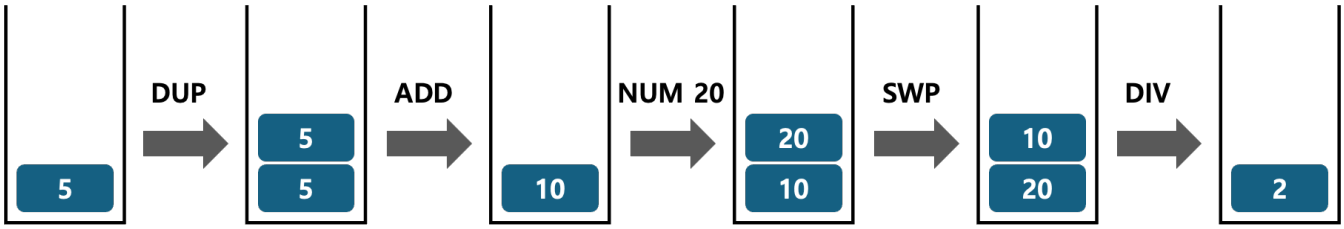
Example 4

Input	Output
5 3.60 4 A+ 3 B+ 3 C+ 1 D0 3	impossible

Problem 2. Stack Calculator

There is a calculator that is based on a slightly modified stack, called a *stack calculator*. The stack calculator can only store numbers and perform the following 10 operations. For convenience, the topmost number in the stack is referred to as the first number, the next as the second number, and so on in order.

- **NUM X**: Push X onto the top of the stack ($0 \leq X \leq 10^9$).
- **POP**: Remove the first number from the stack.
- **INV**: Invert the sign of the first number (e.g., 21 → -21).
- **DUP**: Duplicate the first number and push the copy onto the top of the stack.
- **SWP**: Swap the first and second numbers on the stack.
- **ADD**: Add the first and second numbers.
- **SUB**: Subtract the first number from the second number (second - first).
- **MUL**: Multiply the first and second numbers.
- **DIV**: Divide the second number by the first number and store the quotient. The second number is the dividend, and the first number is the divisor.
- **MOD**: Divide the second number by the first number and store the remainder. The second number is the dividend, and the first number is the divisor.



In the case of binary operators, consider the first number to be the “right” operand, the second number the “left” one. Before performing the operation, both numbers are removed from the stack, and the result is stored back in the stack.

When there are insufficient numbers to perform an operation, when dividing by zero (for DIV and MOD), or when the absolute value of the result exceeds 10^9, it results in a program error. If a program error occurs, the execution of the current program is halted, and no further commands are executed.

To avoid ambiguity in negative division, the calculations are performed as follows: When there is a negative operand in the division, the absolute value of the number is taken before performing the division. Then, the sign of the quotient and remainder is determined as follows: if only one of the operands is negative, the sign of the quotient is negative. In all other cases, the sign of the quotient is positive. The sign of the remainder is the same as the dividend. For example, 13 **DIV** −4=−3, −13 **MOD** 4=−1, and −13 **MOD** −4=−1.

Input

- The input consists of descriptions for multiple calculators. Each calculator’s description is divided into a program section and an input section.
- The **program section** consists of commands, with each command on a separate line. Each command is a three-letter uppercase alphabetic string as specified in the problem description, and no other characters are provided. In the case of the command NUM, a number follows the command, which is

an integer greater than or equal to 0 and less than or equal to 10^9. The NUM and the number are separated by a space. Each program ends with the command END.

- The **input section** begins with the number of program executions, N $(0 \leq N \leq 10{,}000)$. The next N lines each contain one input value V_i $(0 \leq V_i \leq 10^9)$. Each input value is to be executed once by the program, and these executions are all independent. Each time the program is executed, the only value in the stack is the input value V_i.
- Descriptions for each machine are separated by a blank line. The appearance of **QUIT** indicates that there are no further machine descriptions. There will not be more than 100,000 commands, and the stack will not store more than 1,000 numbers during execution.

Output

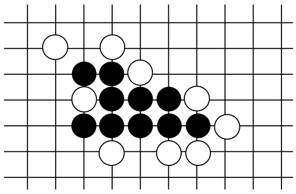
- For each input value, after executing the corresponding program, the output value should be printed. The output value is the number stored in the stack.
- If a program error occurs, or if there is not exactly one number in the stack after all executions are completed, "ERROR" should be printed.
- After printing all output values for each machine, a blank line should be printed.

Example

Input	Output
DUP MUL NUM 2 ADD END 3 1 10 50 NUM 1 NUM 1 ADD END 2 42 43 NUM 600000000 ADD END 3 0 600000000 1 QUIT	3 102 2502 ERROR ERROR 600000000 ERROR 600000001

Problem 3. Omok (a.k.a., Gomoku)

Omok is a game played on a Go board with black and white stones placed alternately. The board has 19 horizontal lines and 19 vertical lines, with the horizontal lines numbered from 1 to 19 from top to bottom, and the vertical lines numbered from 1 to 19 from left to right.



In the above figure, if five consecutive stones of the same color are placed, that color wins. "consecutive" means in the horizontal, vertical, or diagonal direction. Thus, the figure shows a case where the black stones have won. However, if six or more stones are placed consecutively, it does not count as a win.

Given a state of the Go board as input, write a program to determine whether the black stones have won, the white stones have won, or if the game is still undecided. Note that the input will not include cases where both black and white win simultaneously, or where either color wins in two or more places at the same time.

Input

- Each of the 19 lines is represented by 19 numbers, where a black stone is denoted by 1, a white stone by 2, and an empty space is represented by 0. The numbers are displayed with a space between each.

Output

- In the first line, print 1 if black wins, 2 if white wins, or 0 if the game is undecided. If black or white wins, print the row and column number of the leftmost stone among the five consecutive stones (if the five consecutive stones are placed vertically, print the topmost stone) in the second line.

Example

Input	Output
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 2 2 2 1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0	1 3 2